6. XML Schema

This sheet contains exercises about XML Schema.[1] In all the exercises, use the Eclipse "design" view as far as possible to create your schema.

1. Transforming a DTD into an XML Schemaaa

Consider the following DTD specifying the format of documents with information about juicers, that is, devices to obtain juice from fruit:

```
<!ELEMENT juicers (juicer)*>
<!ELEMENT juicer (name, image, description, warranty?,
                  weight?, cost+, retailer)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT image (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT warranty (#PCDATA)>
<!ELEMENT weight (#PCDATA)>
<!ELEMENT cost (#PCDATA)>
<!ELEMENT retailer (#PCDATA)>
```

Convert this DTD into an equivalent XML schema. Do a straight one-to-one conversion (i.e., in the DTD where the elements are declared to contain #PCDATA declare the corresponding element in the XML schema to be of type string). Once you have the schema completed, validate it. Then modify the document `juicers.xml` to indicate that it conforms to your schema. Finally, validate the instance document against your schema.

2. Use More Meaningful Built-In Types

Using a DTD, one can only specifcy the elements with atomic content as containing `#PCDATA`. By translating the DTD into XML schema, we have inserted the built-in type string instead of `#PCDATA`.
XML Schema, however, offers a large number of different built-in types. In this exercise, you are to change the datatypes from string to another built-in datatype, wherever it makes sense. For instance, `weight`, `cost`, and `retailer` can be declared with a much more relevant datatype than string. (Check the document to see how to do this.)
As before, validate the instance document against your schema.

[1] The exercises are based on the labs in the XML Schema tutorial by Roger L. Costello, `http://www.xfront.com/xml-schema.html`

3. Creating a New Datatype by Restriction

Modify the schema that you created in the previous exercise. Create a new datatype called `money` and declare the juicer `cost` element to be of that type.
Hint: Here are the facets for the built-in datatype `decimal`:

`totalDigits`: the total number of digits allowable in the number (including the digits to the right of the decimal point)

`fractionDigits`: the number of digits allowed to the right of the decimal point

`pattern`: regular expression specifying the possible values

`enumeration`: one of the possible values

`whitespace`: instructs the XML processor how to deal with white space; possible values are `preserve`, `replace`, or `collapse`

`maxInclusive`, `maxExclusive`, `minInclusive`, `minExclusive`: self-explanatory

4. Restricting a Datatype Using Regular Expressions

Modify the schema further that you created in Exercise 3. Note that the `image` element contains a string that specifies an image file in the local file system.
Create a new datatype called `imageFile` and declare the `image` element to be of that type.
Define a regular expression for `imageFile`. (Consult the lecture slides to understand how to write regular expression in XML Schema.)
Again, validate the instance document against your schema.

5. Extending Datatypes

Modify the schema that you created in Exercise 4.
Create a type called `appliance`. Define `appliance` to contain declarations for `description` and `warranty`. Create a type called `juiceAppliance`, which is derived from `appliance` (by extension). Make `juicer` of type `juiceAppliance`.
Validate the instance document against your schema.

Note: To satisfy the new schema, the contents of each `juicer` element need to be rearranged in the instance document! For the validation, use `juicers5.xml`, where this has already been done.

6. Schemas with Attributes

Consider now an extension of our original DTD, where the element `juicer` has attributes.

```
<!ELEMENT juicers (juicer)*>
<!ELEMENT juicer (name, image, description, warranty?,
                  weight?, cost+, retailer)>
<!ATTLIST juicer id ID #REQUIRED
                 electric (true|false) #REQUIRED
                 type (press | gear | centrifugal) #REQUIRED>
```

```
<!ELEMENT name (#PCDATA)>
<!ELEMENT image (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT warranty (#PCDATA)>
<!ELEMENT weight (#PCDATA)>
<!ELEMENT cost (#PCDATA)>
<!ELEMENT retailer (#PCDATA)>
```

Create corresponding attributes in the XML schema. Validate the document `juicers6.xml` against the new schema.

7. Extending Simple Types with Attributes

Consider a further extension of the DTD, where the element `cost` has an attribute `currency`.

```
...
<!ELEMENT cost (#PCDATA)>
<!ATTLIST cost currency (USD | CAD) #REQUIRED>
...
```

Create a corresponding attribute in the XML schema. Note that in this case, you have to extend a simple type (`money`) with an attribute.
Validate the document `juicers7.xml` against the new schema.

8. Sets instead of Sequences

In the last schema you defined the content of a `juicer` element as a sequence of other elements. There is no inherent reason, however, that the children of the juicer element should occur in a specified order. Change the schema so that the juicer element contains the same set of child elements but allow them to occur in any order.

Hint 1: (1) Whenever a type extends a base type, the extension elements are always appended to the base type's elements:

$$b_1, b_2, e_1, e_2$$

where b_1 and b_2 are the base type's elements and e_1, e_2 are the elements from the type that is extending the base type.
Thus, even if one makes the base type unordered and the type extending the base type unordered, the resulting set of elements will still be partially ordered as

$$\{ b_1, b_2 \}, \{ e_1, e_2 \}$$

where the curly braces indicates an unordered set. We have an unordered first set followed (sequentially) by an unordered second set.
So, for this exercise you are to delete the `appliance` type and place its element declarations in with the `juiceAppliance` type.

Hint 2: One of the restrictions on using the `<all>` element is that the elements declared within `<all>` must have a `maxOccurs="1"`. So, remove the cost that specifies the cost in Canadian Dollars. (We will just deal in US currency.)

The file `juicers8.xml` contains `juicer` elements in arbitrary order. Validate (1) your new schema and (2) validate the `juicers8.xml` against the new schema.

9. Empty Elements

Reuse again the schema `juicers0.xsd`, which has the appliance complexType. Currently the retailer element has as its content a URI. Modify it so that its content is empty and it instead has an attribute—`href`—whose type is `anyURI`.

The file `juicers9.xml` has empty retailer elements. Validate the instance document against your schema.

10. Creating an Appliances Repository Schema

Modify the XML file and the schema that you created in the last exercise.
In all of the exercises thus far we have stored all of our schema components in a single schema. Typically, however, one will put elements and types that may be used by many schemas into a separate schema (a repository schema).
In our juicer example the appliance type is an example of something that could be used in many schemas. Hence, pull it out of `juicers.xsd` and put it into a generic schema, `appliances.xsd`. Then modify `juicers.xsd` to use the definition of appliances in `appliances.xsd`.
Use the same namespace for the two schemas.

11. Demonstating the Chameleon Effect

Modify the `appliances.xsd` schema that you created in Exercise 10 to have no targetNamespace.

12. Using Multiple Namespaces

Modify again the juicers file and the schemas that you created in Exercise 10. Put the `appliances.xsd` schema in its own namespace (`http://www.appliances.org`). Modify `juicers.xsd` to use the `appliances` complexType, which is now in a different namespace. You will need to modify the instance document to reflect the fact that it is using multiple namespaces.

13. Creating Lists

Modify the juicers file and the schemas that you created in Exercise 12. Assume there are multiple retailers for each juicer. Modify `juicers.xsd` to allow for a list of values for the `href` in the retailers element.

14. Declaring Key and Uniqueness Constraints

Take again `juicers0.xsd` and `juicers0.xml` as a starting point.
Add the constraints that for juicers, the `name` attribute is a key and that values of `image` are unique. Introduce errors into the XML file to check whether they are detected by the validator.